Nek Deep Dive

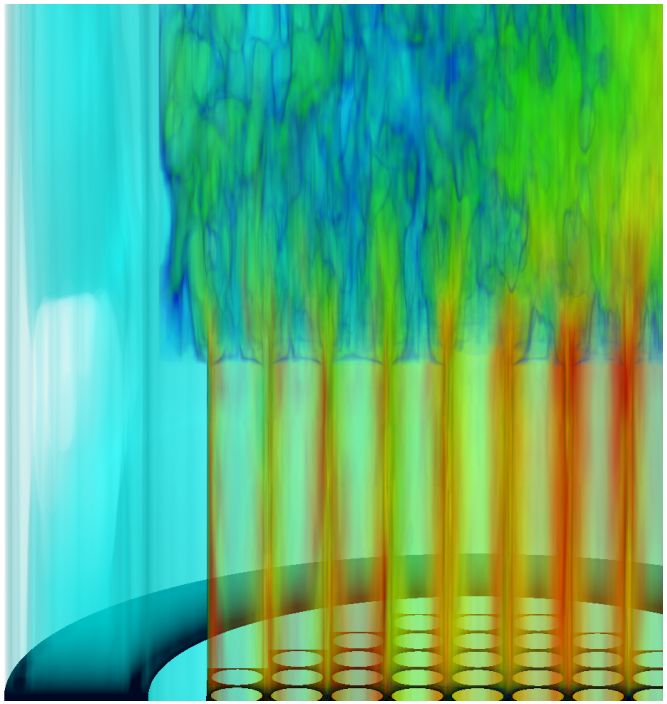
E. Merzari, K. Heisey, P.F. Fischer

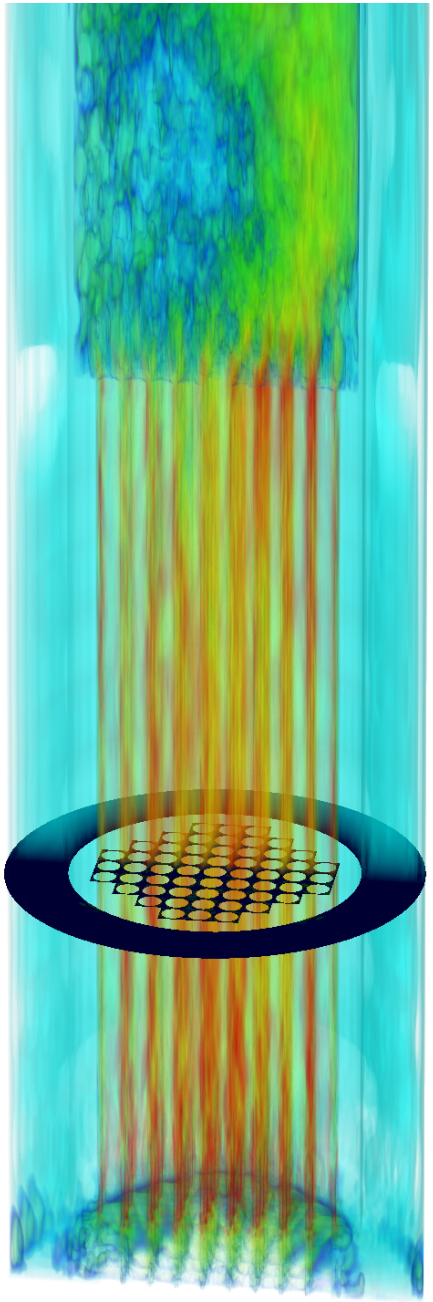
Nuclear Engineering Division
Argonne National Laboratory
&
Mathematics and Computation Division
Argonne National Laboratory

9700 S. Cass Avenue, Lemont, IL 60439

Outline

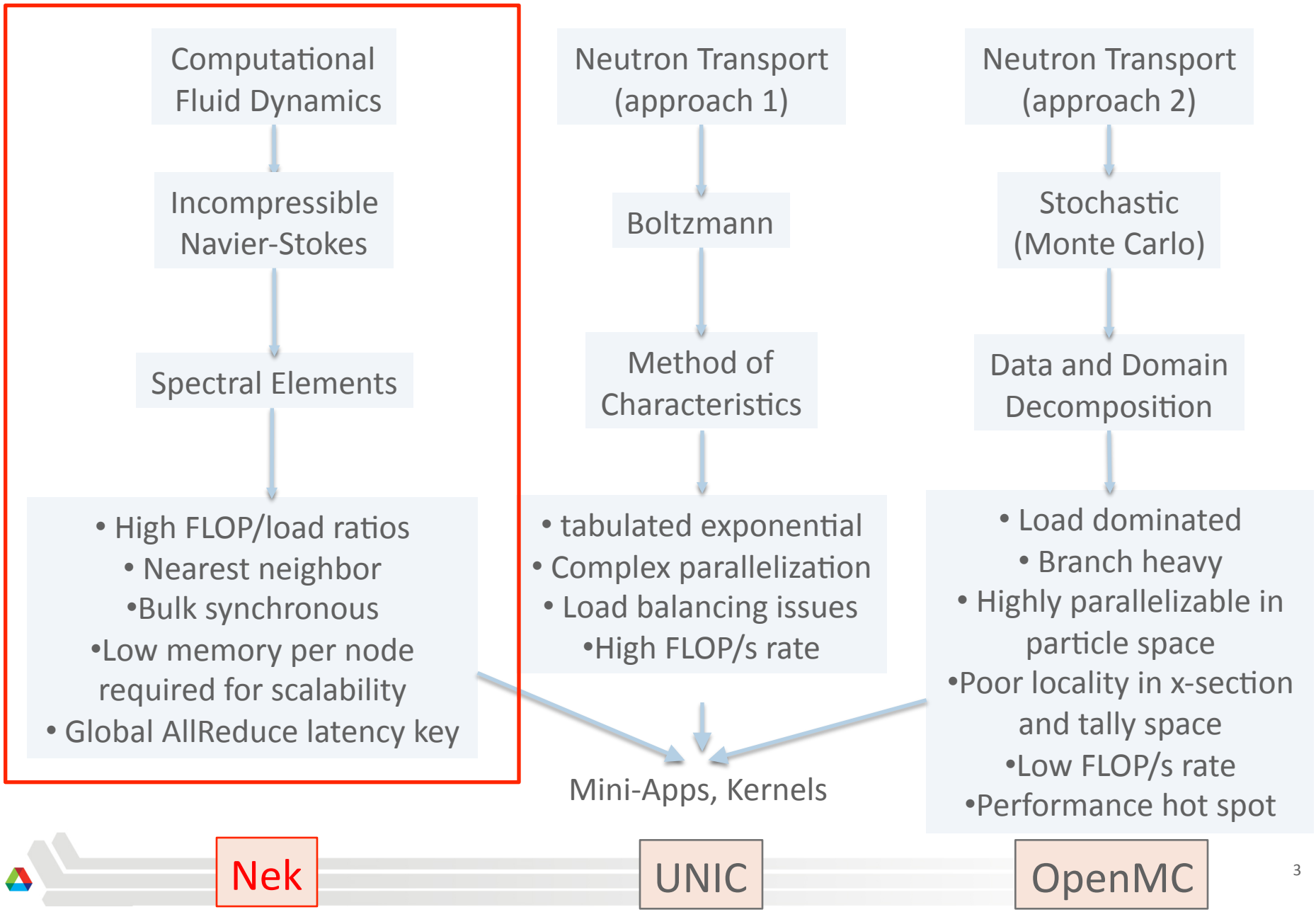
- Background (10 min)
  - Navier-Stokes equations
  - Why high-order for turbulence
  - “High-order accuracy at low-order cost”
  - Overview of the N-S Solver
- Nekbone App (15 min)
  - Overview
  - Current tests
- Overview of Methods (25 min)
  - SEM
  - Pressure Solver in Nek
  - Nekbone/Nek differences
  - Scaling issues





Three-dimensional rendering of a Nek5000 simulation of the MASLWR experiment

Three codes are focus of CESAR research



Nek - Principal Application

Incompressible Navier-Stokes equations

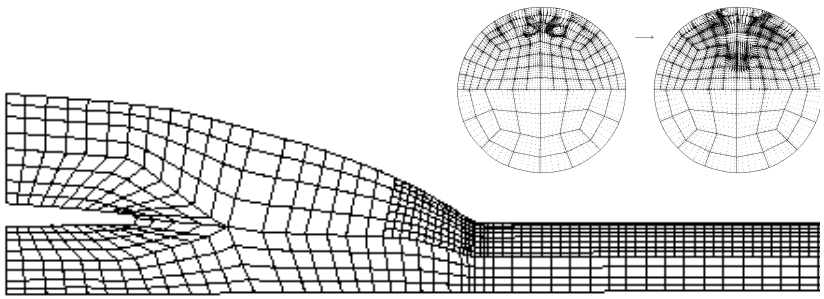
$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u}$$

$$\nabla \cdot \mathbf{u} = 0$$

- **Reynolds number Re > ~1000**
  - *small amount of diffusion*
  - *highly nonlinear* (small scale structures result, range of scales widens for increasing Re)
- Excellent Approximation for a variety of reactor problems: Incompressibility is an excellent approximation to single-phase reactor flows.
- **Very General Applicability**: Many systems of engineering interest (ex. Combustion Engines, Heat Exchangers), Blood Flow, etc..

Spectral Element Method & Transport Problems

- Variational method, similar to Finite Element, using *GL* quadrature.
- Domain partitioned into E high-order quadrilateral (or hexahedral) elements (decomposition may be nonconforming - *localized refinement*) .Converges *exponentially fast* with N
- Trial and test functions represented as N th-order tensor-product polynomials within each element. (N : 4 -- 15, typ., but N = 1-100+ also works.) $n \sim EN^3$ gridpoints in 3D, EN^2 gridpoints in 2D.

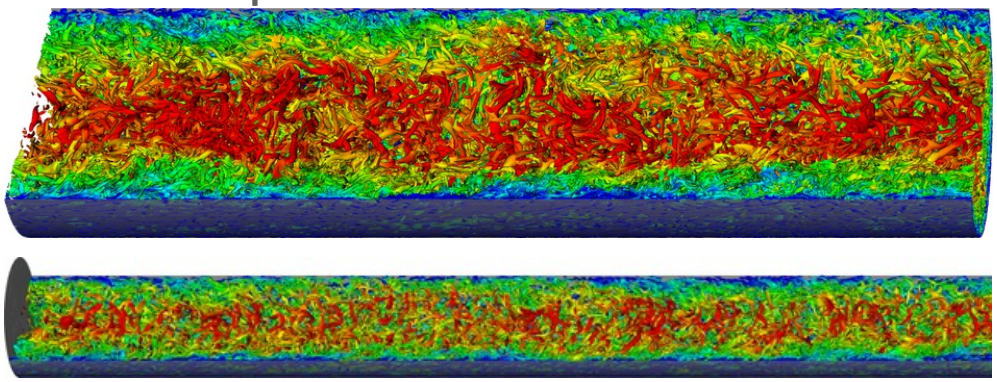


- These problems are particularly challenging because, unlike diffusion, where:

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u \quad \longrightarrow \quad \hat{u}_k(t) \sim e^{-\nu k^2 t}$$

implies rapid decay of high wavenumber (k) components (and errors), the high-k components and errors in advection-dominated problems *persist.*

- Turbulence provides a classic example of this phenomena, Excellent performance of SEM. **High-order methods can efficiently deliver small dispersion errors.**

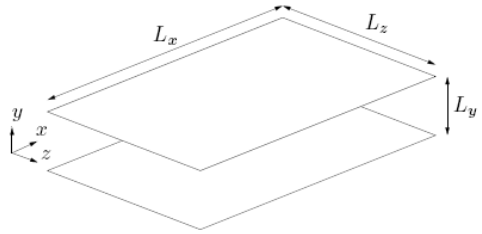


Spectral element simulations of turbulent pipe flow

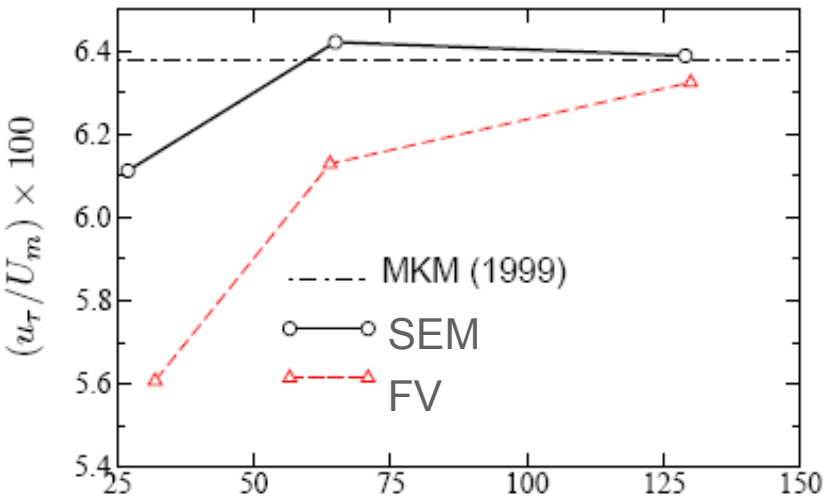
Benefit of High Order Methods

From Sprague et al., 2010

Comparison for DNS in channel flow: standard test case in turbulence research

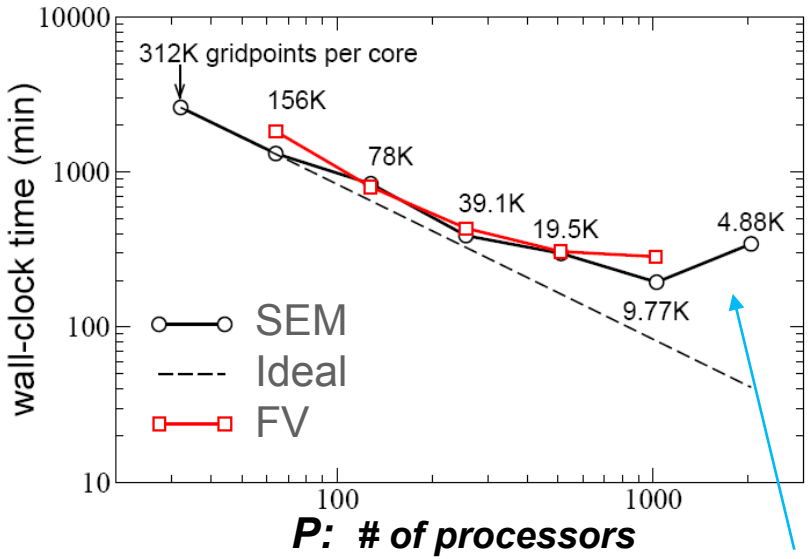


Accuracy



n_y: # of points in wall-normal direction

Performance



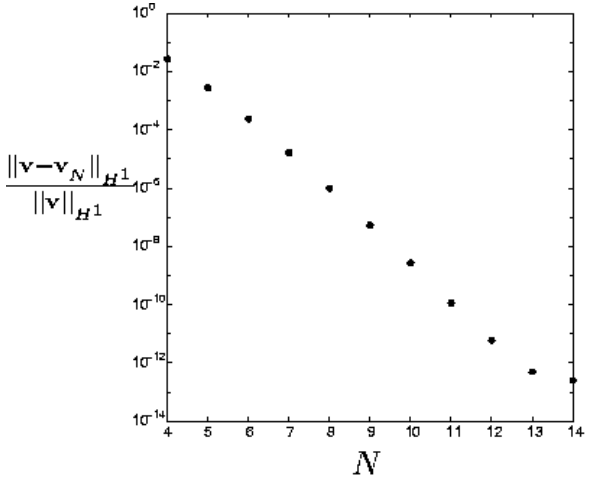
***P*: # of processors**

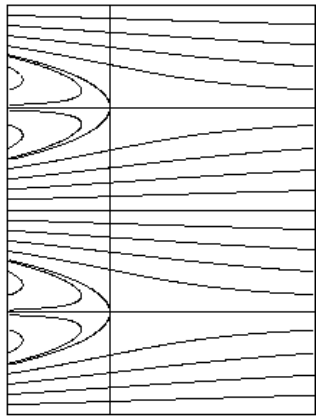
Roll-over at n/P ~ 10K

- — SEM & FV have the *same* cost per gridpoint
- — For same accuracy, over several metrics, **FV needs 8-10 times as many points as 7th-order SEM**

Remember: These are useful flops!

- *Exponential convergence:*
  - *e.g., doubling the number of points in each direction leads to 10^4 reduction in error, vs. 4x for 2^{nd}-order schemes.*
  - *Minimal numerical dispersion –* ***essential for exascale problems.***





$$
\begin{aligned}
v_x &= 1 - e^{\lambda x}\cos 2\pi y \\
v_y &= \frac{\lambda}{2\pi} e^{\lambda x}\sin 2\pi y \\
\lambda &:= \frac{Re}{2} - \sqrt{\frac{Re^2}{4} + 4\pi^2}
\end{aligned}
$$

Exact Navier-Stokes solution, Kovazsnay(1948)

Overview of the Navier-Stokes Solver in Nek

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2\mathbf{u}$$
$$\nabla\cdot\mathbf{u} = 0$$

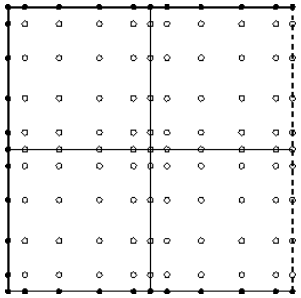
$$\begin{bmatrix} \mathbf{H} & -\frac{\Delta t}{\beta_0}\mathbf{H}\mathbf{B}^{-1}\mathbf{D}^T \\ \mathbf{0} & E \end{bmatrix} \begin{pmatrix} \underline{\mathbf{u}}^n \\ \underline{p}^n - \underline{p}^{n-1} \end{pmatrix} = \begin{pmatrix} \mathbf{B}\underline{\mathbf{f}} + \mathbf{D}^T\underline{p}^{n-1} \\ \underline{g} \end{pmatrix} + \begin{pmatrix} \underline{\mathbf{r}} \\ \underline{0} \end{pmatrix} \quad ,$$

$$E := \frac{\Delta t}{\beta_0}\mathbf{D}\mathbf{B}^{-1}\mathbf{D}^T \quad , \qquad \underline{\mathbf{r}} = O(\Delta t^2)$$

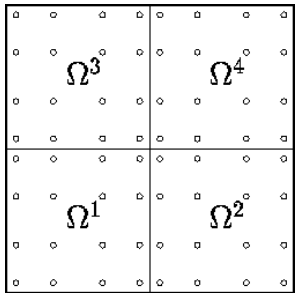
- Semi implicit treatment of the nonlinear term: *explicit* (k th-order backward difference formula / extrapolation (k =2 or 3) - k th-order characteristics)
- Leads to linear Stokes problem: pressure/viscous decoupling:
  - 3 Helmholtz solves for velocity *("easy" w/ Jacobi-precond.CG)*
  - **(consistent) Poisson equation for pressure** *[calls Helmholtz solver]*
  - - SPD, Stiffest substep in Navier-Stokes time advancement, Most compute-intensive phase
  - Spectrally equivalent to SEM Laplacian, A
- Additional operations to ensure stability and turbulence resolution

Velocity, **u** in P_N , continuous
Pressure, p in P_{N-2} , discontinuous *or*

Velocity, **u** in P_N , continuous
Pressure, p in P_N , discontinuous

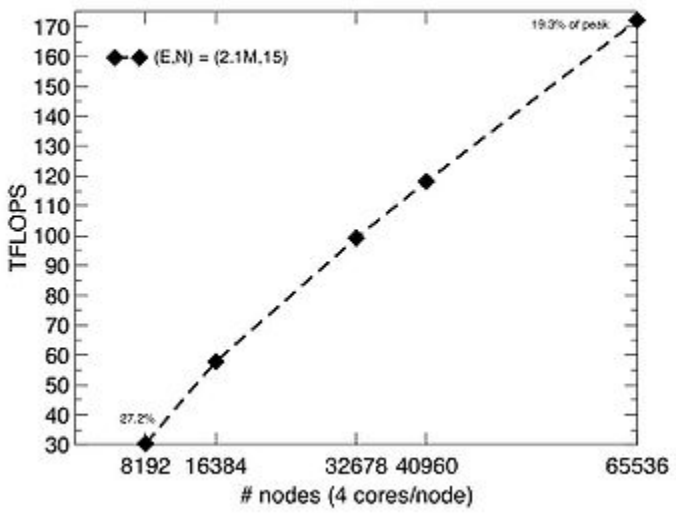


Gauss-Lobatto Legendre points (P_N)

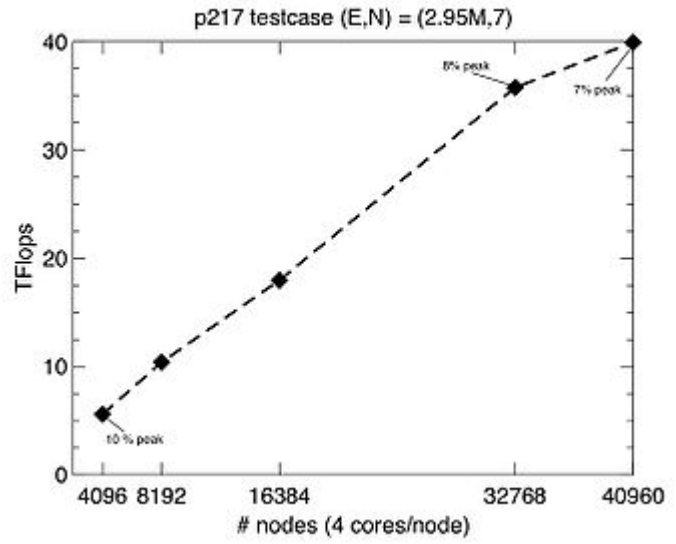


Gauss Legendre points (P_{N-2})

Strong Scaling: Where We Are



- [Juelich's IBM BG/P Eugene (32768-262144 cores)]
- 2.1M spectral elements with a polynomial order of N=15
- fixed velocity/pressure iterations 5/50
- full scale parallel efficiency = 0.71
- peak application performance = 172.1TFlops (19.3% of peak)
- elapsed time / timestep / gridpoint = 483 us (averaged over 20 timesteps)

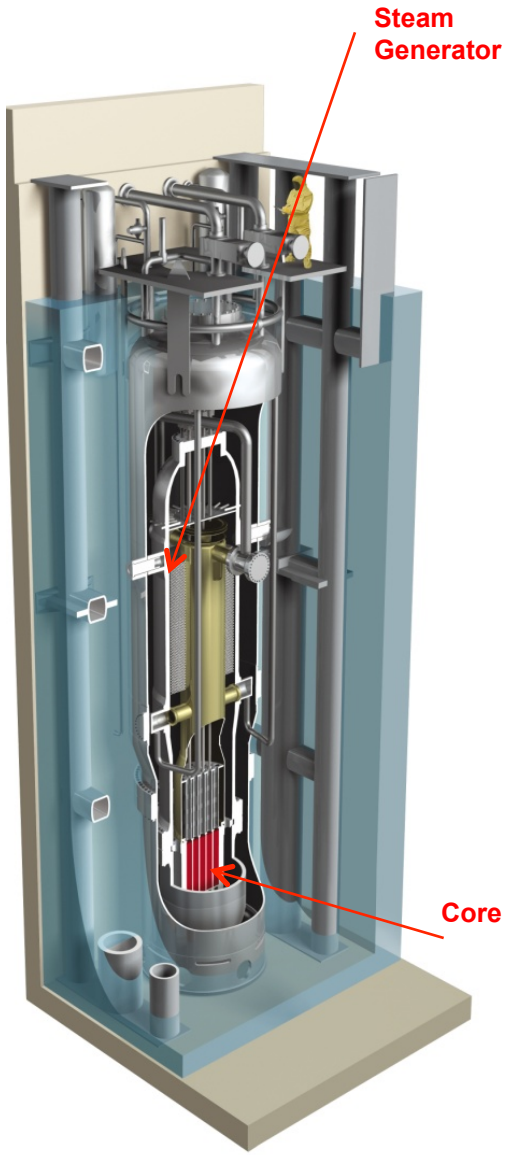


- [ANL's IBM BG/P Intrepid (16384-163840 cores)]
- 2.95M spectral elements with a polynomial order of N=7
- fixed velocity/pressure iterations 8/20
- full scale parallel efficiency = 0.7
- peak application performance = 40.1TFlops (7.2% of peak)
- elapsed time / timestep / gridpoint = 96 us (averaged over 20 timesteps)

- [Recently we were able to run with 10^6 MPI processes]

A Realistic Reactor Problem (Exascale)

- *Nek is not an App.* *It is a code.* *An App is when you apply Nek to a specific problem.*
- It makes no sense to consider solving most of current Petascale problems on an exascale machine. Reactor Problems are ideal for Exascale.
- Larger system (increased number of channels) but at SCALE (same a-dimensional numbers) with available experiments.
- The computational cost can be predicted in a straightforward manner using Petascale simulations of at SCALE facilities.
- ~ trillion points (tens of billion CPU hours needed in the best case)
- Exascale might also help perform parametric studies (hundreds of current petascale simulations) – e.g. to help study the effect of the heating rate on stability.



Nekbone

Nekbone: Introduction - 1

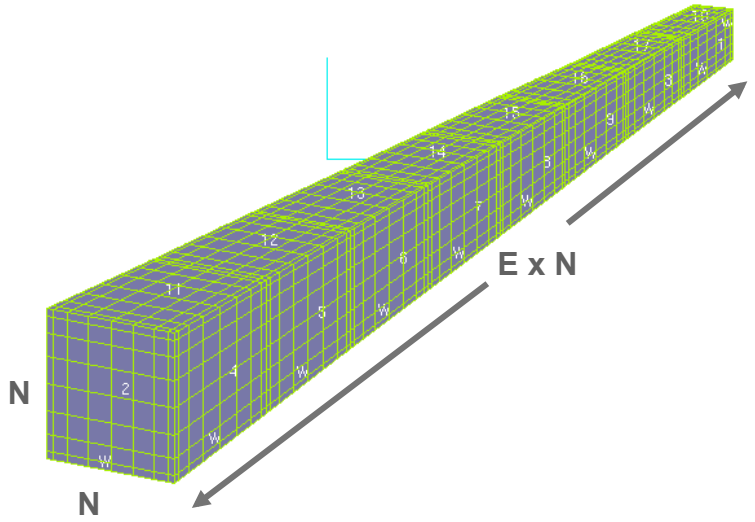
- *Goal:*
  - ***Provide light-weight code to capture the principal hot kernel in Nek.***
  - *Elliptic solver:*

$$H\underline{u}^n := (I + \Delta t A)\,\underline{u}^n = \underline{f}^n$$

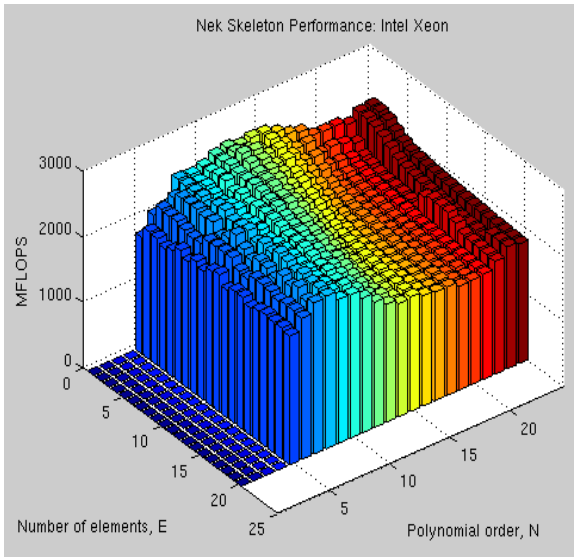
- **Solves Helmholtz equation in a box using the spectral element method**. Currently, box is a 1D array of 3D elements or a box of 3D elements.
  - Design decision: simple, rather than many options.
  - Semi-Implicit formulation: Evaluation of a Poisson equation at every time step – most of computational cost in Nek is in the Poisson solve (**62% of a representative run**)
  - Also: **any Nek5000 application that spends a majority of time in the temperature solver will very closely resemble a Nekbone test ran** on a large, brick element count.
- Provides estimates of realizable flops, as well as internode latency and bandwidth measures.
- ***Diagonally-preconditioned CG iteration*** *to solve the Poisson equation*
  - *Different from Nek : Optimized Two level preconditioner in Nek vs. Diagonal preconditioner in Nekbone*

Nekbone : Introduction -2

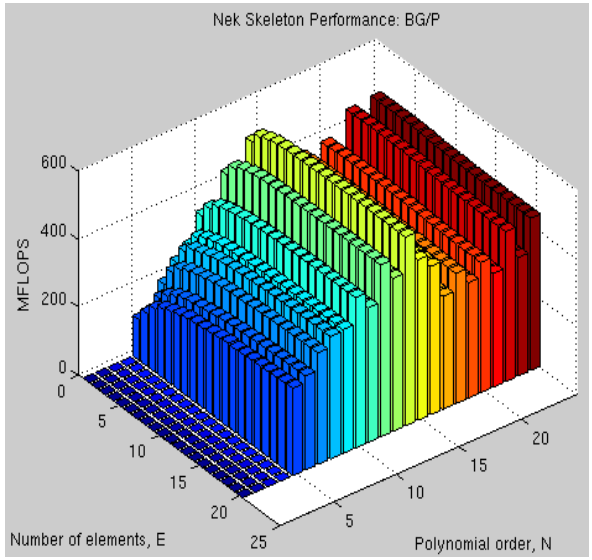
- Current Focus on diagonally-preconditioned CG iteration, involving [More on this later]:
  - Operator-evaluation: $\underline{w} = A\,\underline{p}$
  - Nearest-neighbor communication: $\underline{w} = QQ^T\,\underline{w}^*$
  - All-reduce: $a = \underline{w}^T\underline{p}$
- $A\,\underline{p}$ kernel offers several opportunities for parallelism to be explored.
- Essentially : many matrix-matrix products
- Tests run through a battery of problem sizes, $n = P \times E \times N^3$
  - P = number of cores
  - E = number of blocks (spectral elements) per core
  - N= block-size (polynomial order in each direction)
  - Run on Intel, BG/P, and BG/Q



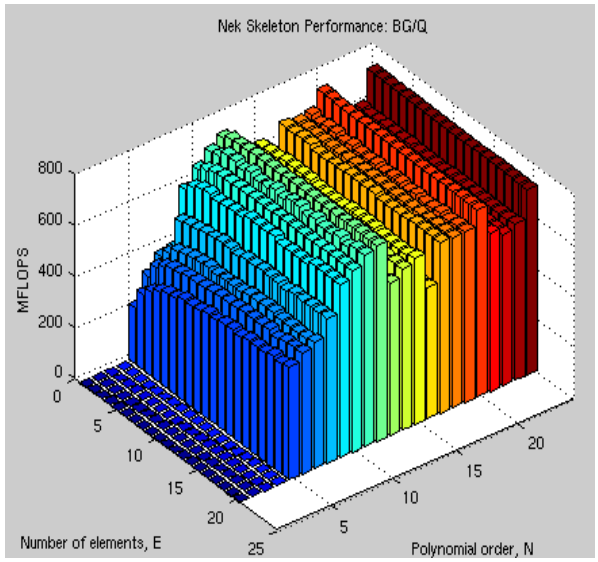
Nekbone: Initial Results on Xeon, BG/P, and BG/Q



Xeon



BG/P



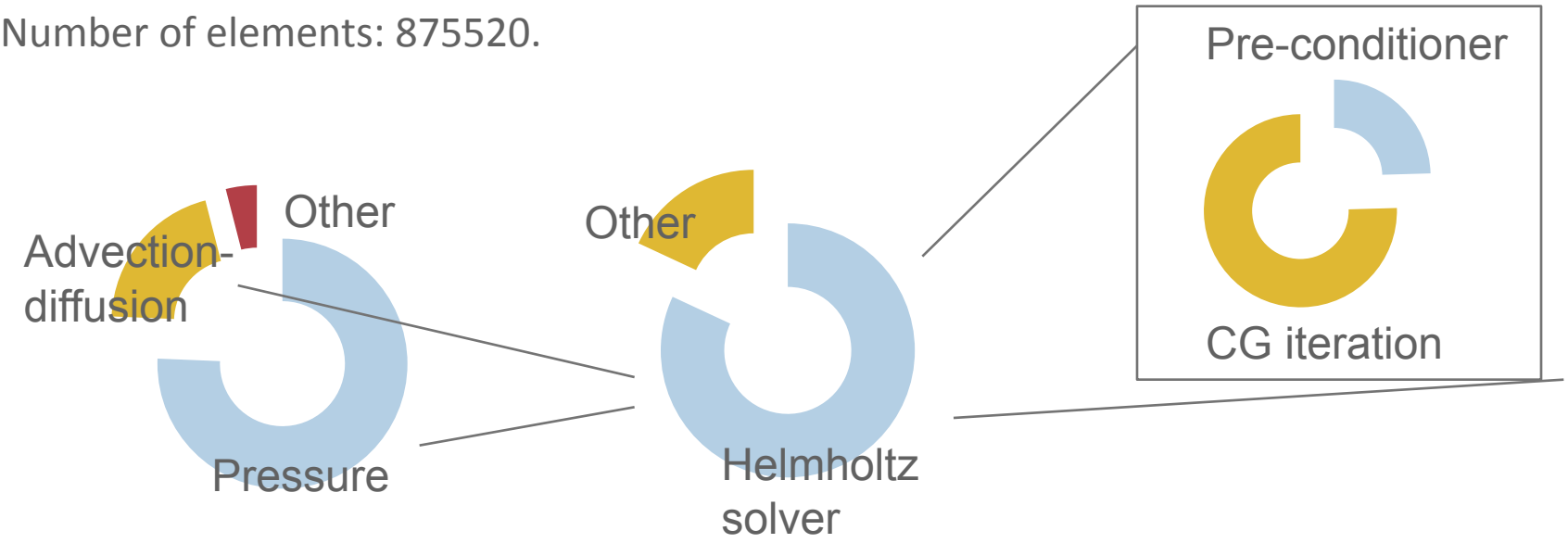
BG/Q

- MFLOPS/core for full skeleton kernel, 400 tests in (E,N) space.
- Xeon shows cache dependencies (less data = more performance)
- BG/P and Q show less cache sensitivity.
- Max-performance block size (N):

  Xeon: N=8 (2.3 GFLOPS), BG/P: N=14 (.55 GFLOPS), BG/Q: N=12 (.765 GFLOPS)

Representative Nek case vs. Nekbone

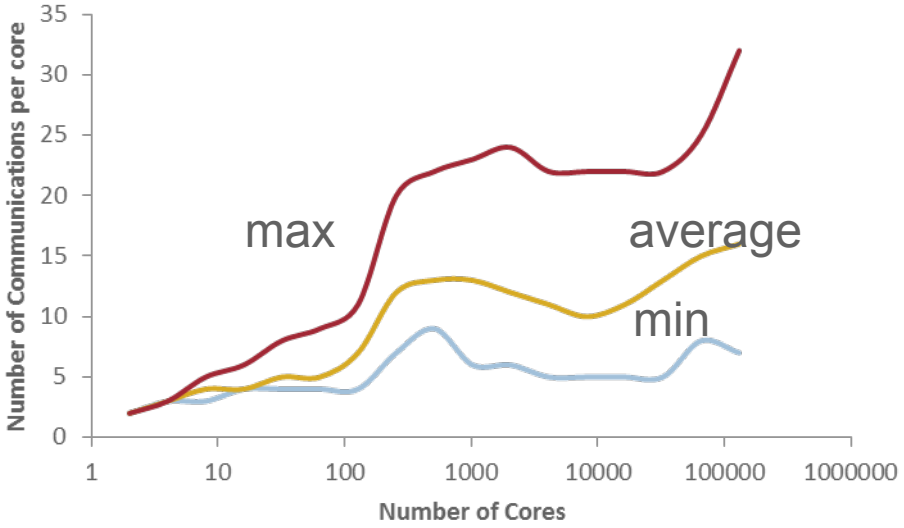
- The case is a natural convection case at high Rayleigh number (Ra > 10^{10}).
- Number of processors: 32768.
- Number of elements: 875520.

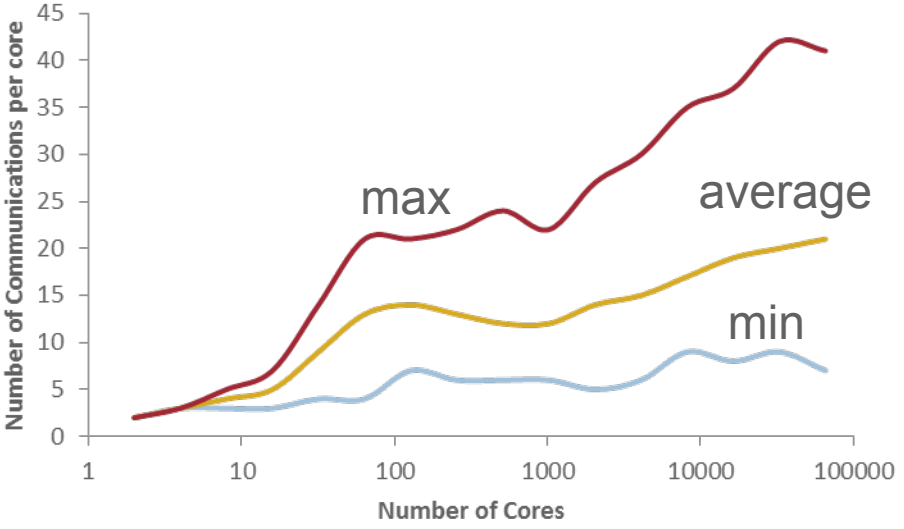


- From this example, we see that the Helmholtz solve consumes 82% of the CPU time (75% in pressure, 7% in u,v,w, and T).
- Of this, 20% is spent in the pre-conditioner, [which is not yet in nekbone, but will be, once the nekbone users are ready for it]
- It makes more sense, however, to focus on the 63% of the time that is represented by the fairly simple nekbone kernel (essentially the CG iteration).
- **REPRESENTATIVE OF THE ENVISIONED APPLICATION**

Communication: Nek vs. Nekbone

- ***Nearest neighbor Communication: Source of Load Imbalance***
- ***Case-dependent!***
- One element NxNxN per core: 8 messages of size 1; 12 messages of size N; and 6 messages of size N^2
- Message volume is dominated by the 6 face exchanges.
- At fine granularity (latency dominates), each of this exchanges has a similar cost(depends on whether is exceeds m_2)
- Nekbone has two modes of operation:
  - a string of elements in a 1 x 1 x E array (**communication-minimal**)
  - a block of elments E1 x E2 x E3, which has 26 neighbors (**representative**).

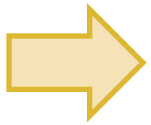




Complexity: Nek vs. Nekbone

How we plan to increase mini-app fidelity/complexity:

Where we are now



Diagonally-preconditioned CG

Addition of Schwarz smoother
increase in comm and syncronicity

Addition of coarse grid solve
significant increase in global comm and syncronicity

Switch to GMRES
increase in local memory traffic w/o much work

Nekbone - Summary

The principal challenge at exascale is going to be to boost or retain reasonable single-node performance. To leading order, this question is ***NOT*** a scaling question, it is a "hot kernel" question.

Nekbone focuses on the hot kernel

This high-level kernel is simplified, so that it's accessible to computer scientists (it's not the full app. after all).

In terms of flops and fidelity to data flow, it captures the essential points in the smallest piece of code.

It includes access to two essential types of communication

It provides opportunity for optimization at multiple levels. From low to high, these are:

- mxm kernels (already heavily optimized for most platforms)
- gradient kernels (3 mxms on the same data)
- grad-transpose (3 mxms on different data, producing one output)
- operator level - calls to grad
- solver level:
  - calls to operator
  - calls to vector-vector ops
  - calls to all-reduce
  - **calls to gs**.

Methods Overview

A little more about the SEM (Nek & Nekbone)

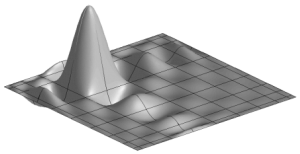
- Nodal basis:

$$u(x,y)|_{\Omega^e} = \sum_{i=0}^{N}\sum_{j=0}^{N} u_{ij}^e\, h_i(r)\, h_j(s)$$

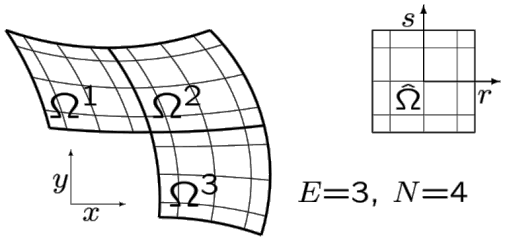
$$h_i(r) \in \mathcal{P}_N(r), \qquad h_i(\xi_j) = \delta_{ij}$$

- ξ_j = Gauss-Lobatto-Legendre quadrature points:

  - stability (*not* uniformly distributed points)
  - allows pointwise quadrature (for *most* operators…)
  - easy to implement BCs and C^0 continuity



2D basis function, N=10



E=3, N=4

- *Local tensor-product form* (2D), *allows derivatives to be evaluated as matrix-matrix products:*

$$\left.\frac{\partial u}{\partial r}\right|_{\xi_i,\xi_j} = \sum_{p=0}^{N} u_{pj} \left.\frac{dh_p}{dr}\right|_{\xi_i} = \sum_p \hat{D}_{ip} u_{pj} =: D_r \underline{u}$$

- *Memory access scales only as O(n) (even in general domains)*
- *Work scales as N*O(n), but CPU time is weakly dependent on N*
- *Tensor Product: "Matrix Free"*
- *For the Stifness Matrix A evaluations:*
  1. *Operation count is O (N 4)*
  2. *Memory access is 7 x number of points*
  3. *Work is dominated by (fast) matrix-matrix products involving Dr , Ds , etc.*

Evaluation of a(v,u) in R^3

In 3D, i $\mathcal{I}$ is given by,

$$\begin{aligned}\mathcal{I} \approx a_N(v,u) &= \underline{v}^T \begin{pmatrix} D_r \\ D_s \\ D_t \end{pmatrix}^T \begin{bmatrix} G_{11} & G_{12} & G_{13} \\ G_{12} & G_{22} & G_{23} \\ G_{13} & G_{23} & G_{33} \end{bmatrix} \begin{pmatrix} D_r \\ D_s \\ D_t \end{pmatrix} \underline{u} \\ &= \underline{v}^T A \underline{u},\end{aligned}$$

with

$$(G_{ij})_{lmn} := \rho_l \rho_m \rho_n \, J_{lmn} \sum_{k=1}^{3} \left(\frac{\partial r_i}{\partial x_k} \frac{\partial r_j}{\partial x_k} \right)_{mnl} .$$

- Look at the *memory access costs:* – only $7(N+1)^3$ to evaluate $A\underline{u}$.
- However, if we *store* A, the cost is $(N+1)^6$! (per element!)
- Recall, there are now $(N+1)^3$ unknowns in $\underline{u}$, or in $\underline{u}^e$ in the multi-element case.

SEM Operator Evaluation (Nekbone & Nek)

- Spectral element coefficients stored on element basis ($\underline{u}_L$ not $\underline{u}$)

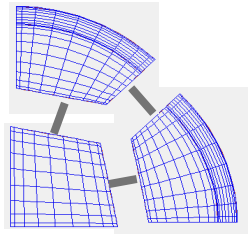
$$\underline{w} = A\underline{u} = Q^T A_L Q\underline{u}, \qquad \underline{w}_L := Q\underline{w}, \qquad \underline{u}_L := Q\underline{u}$$

$$\underline{w}_L = QQ^T A_L \underline{u}_L$$

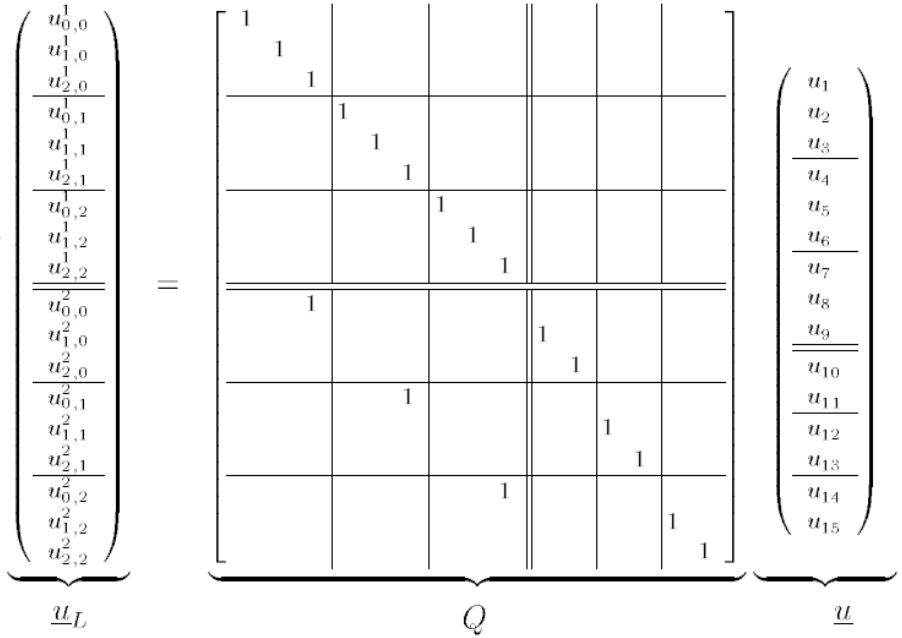
local work (tensor products)

nearest-neighbor (gather-scatter) exchange

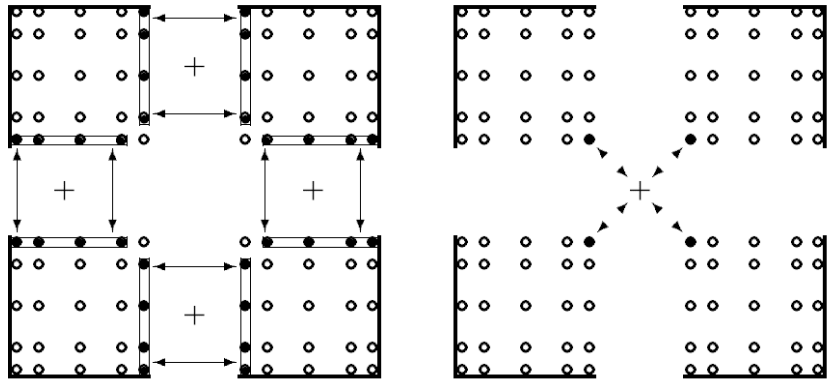
$$A_L := \begin{bmatrix} A^1 & & & \\ & A^2 & & \\ & & \ddots & \\ & & & A^E \end{bmatrix}$$



- Decouples complex physics –and computation - (A_L) from communication (QQ^T)
- Communication is required, and the communication pattern must be established a priori (for performance): set-up phase, gs_setup(), and excecute phase, gs()



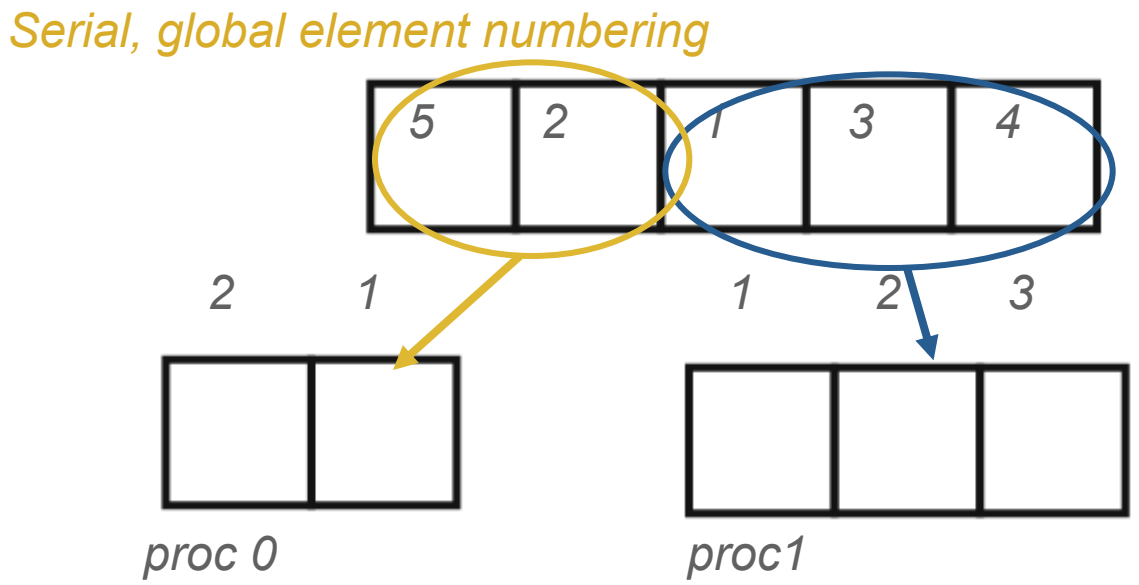
Example of Q for two elements



QQ^T Pictorially

Parallel Structure (Nek & Nekbone)

- Elements are assigned in ascending order to each processor



Communication Kernel:
General Purpose Gather-Scatter (Nek & Nekbone)

- Handled in an abstract way, simple interface.

- Given index sets:

  proc 0: *global_num* = { 1, 9, 7, 2, 5, 1, 8 }

  proc 1: *global_num* = { 2, 1, 3, 4, 6, 10, 11, 12, 15 }

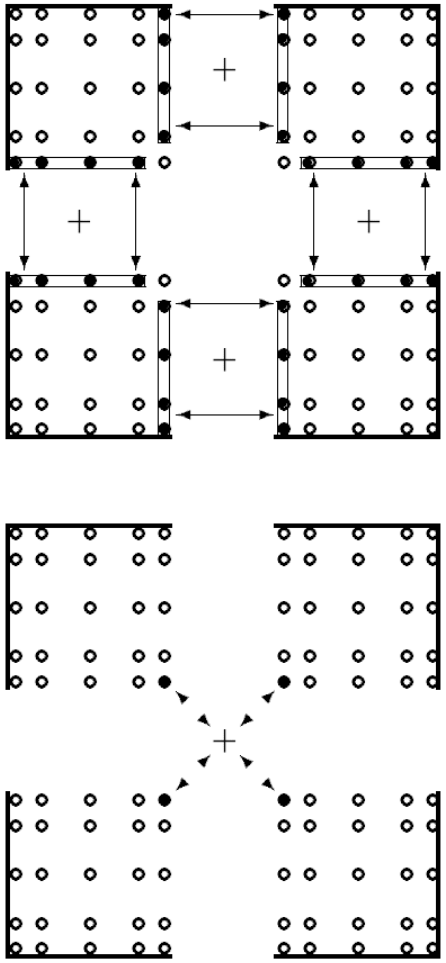
  On each processor: ***gs_handle = gs_setup(global_num,n,comm)***

- In an *execute()* phase, exchange and sum:

  proc 0: u = { u_1, u_9, u_7, u_2, u_5, u_1, u_8 }

  proc 1: u = { u_2, u_1, u_3, u_4, u_6, u_{10}, u_{11}, u_{12}, u_{15} }

  On each processor: ***call gs(u,gs_handle)***

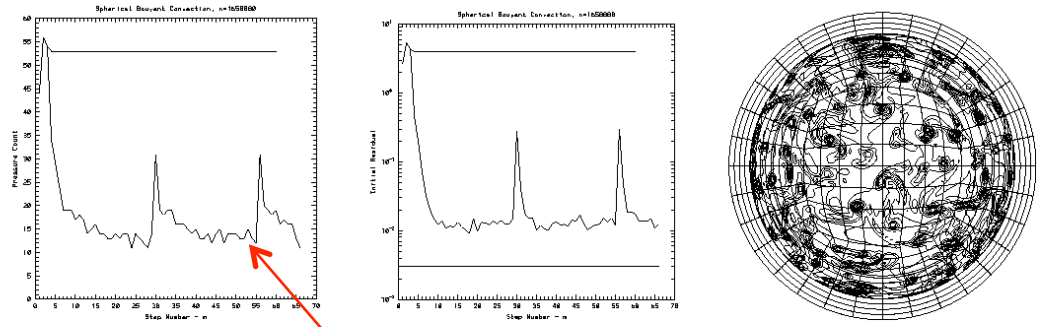


Pressure Solution Strategy (Nek ONLY)

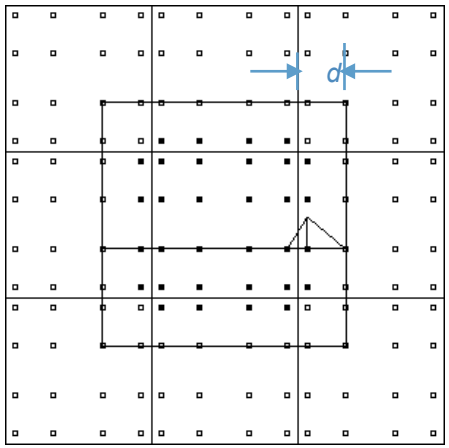
1. *Projection: compute best approximation from previous time steps*
   - *Compute* $\underline{p}^*$ *in span*$\{\underline{p}^{n-1}, \underline{p}^{n-2}, \ldots, \underline{p}^{n-l}\}$ *through straightforward projection.*
   - *Typically a 2-fold savings in Navier-Stokes solution time.*
   - *Cost: 1 (or 2) matvecs in* E *per timestep*
3. *Preconditioned CG or GMRES to solve*

$$E\,\Delta\underline{p} = \underline{g}^n - E\,\underline{p}^*$$

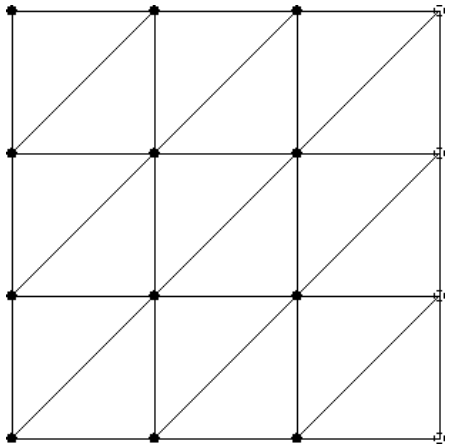
Preconditioning is divided in two steps.



- 4 fold reduction in iteration count, 2 – 4 in typical applications



***Overlapping Solves**: Poisson problems with homogeneous Dirichlet bcs.*



***Coarse Grid Solve**: Poisson problem using linear finite elements on spectral element mesh (GLOBAL).*

Overlapping Schwarz Preconditioning for Pressure (Nek ONLY)

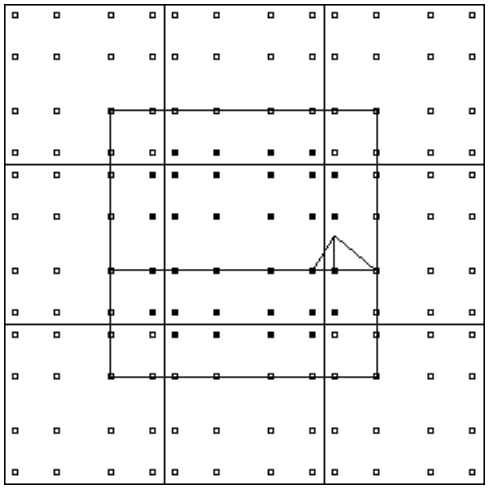
$$\underline{z} = P^{-1}\,\underline{r} = R_0^T A_0^{-1} R_0\,\underline{r} + \sum_{e=1}^{E} R_{o,e}^T A_{o,e}^{-1} R_{o,e}\,\underline{r}$$

$A_{o,e}$ - low-order FEM Laplacian stiffness matrix on overlapping domain for each spectral element k (Orszag, Canuto & Quarteroni, Deville & Mund, Casarin)

$R_{o,e}$ - Boolean restriction matrix enumerating nodes within overlapping domain e

A_0 - FEM Laplacian stiffness matrix on coarse mesh (~ E x E)

R_0^T - Interpolation matrix from coarse to fine mesh



2D: $A = (B_y \otimes A_x + A_y \otimes B_x), \qquad S^T A S = \Lambda, \quad S^T B S = I.$

$$A^{-1} = (S_y \otimes S_x)\,(I \otimes \Lambda_x + \Lambda_y \otimes I)^{-1}\,(S_y^T \otimes S_x^T).$$

NOTE: B_x, B_y, *lumped* 1D mass matrices (conditioning)

Op. Count: $W = 8KN^3$ (*vs.* $4KN^3$ for band solve)

Storage: $S = O(KN^2)$ (*vs.* KN^3 for band solve)

NOTE: $S_y \otimes S_x\,\underline{u} = S_x U S_y^T$ (matrix-matrix product)

- Local and nearest neighbors
- **Exploit local tensor-product structure**
- Fast diagonalization method (FDM) - local solve cost is ~ $4d\,K\,N^{(d+1)}$ (Lynch et al 64)

26

Coarse-Grid Solver (Nek ONLY) - 1

Designed for rapid *solution* performance – preprocessing cost not a constraint.

- Uses coarse/fine (C-F) AMG
- Energy minimal coarse-to-fine interpolation weights
- Communication exploits *gs()* library

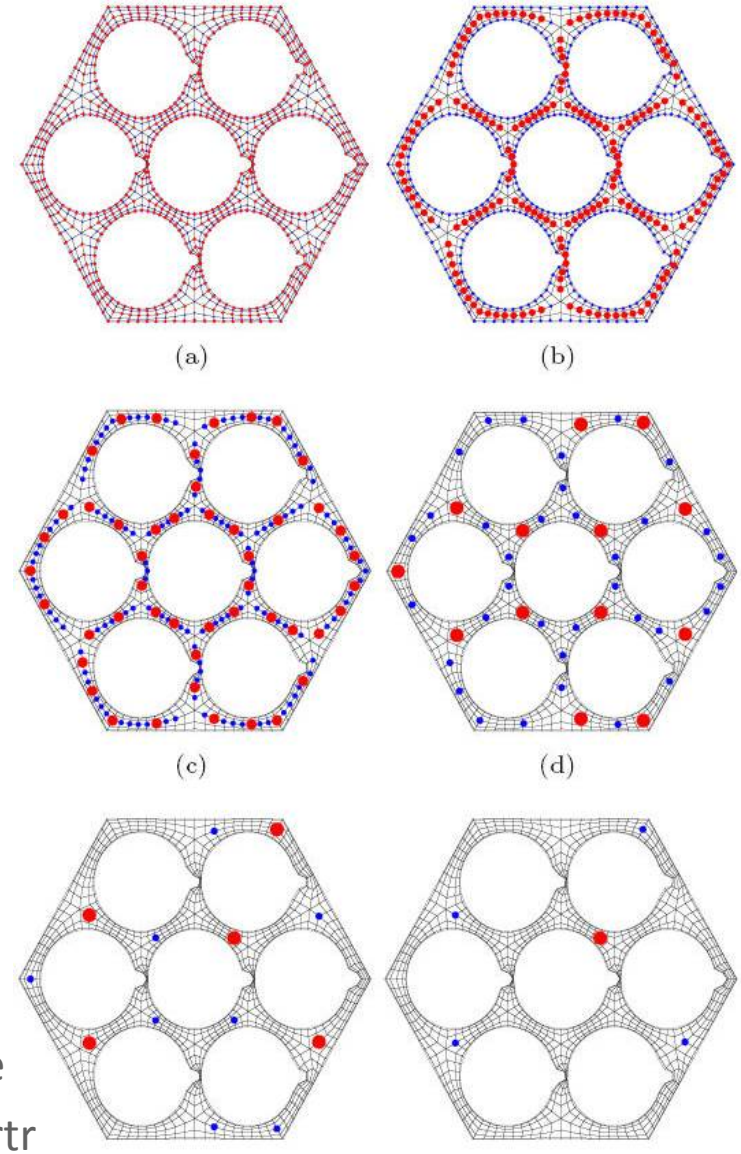
Break-Down of Navier-Stokes Solution Time for n=120 M, n_c = 417000

Case/P	Total	QQ^T	Coarse	all_reduce()
x4096	1994	125	1180	1.2
a4096	1112	125	192	1.4
b4096	846	126	25	1.
8192	460	88	22	1.
16384	266	64	20	1.

XX^T is a fast parallel coarse solver, AMG is necessary

Pairwise+all_reduce

Pwise/all_red/crys.rtr



coarse (red) and fine (blue) points

AMG Statistics for n_c = 417600

Communication stencil width (~ nnz/n_c or nnz/n_f) grows at coarse levels.

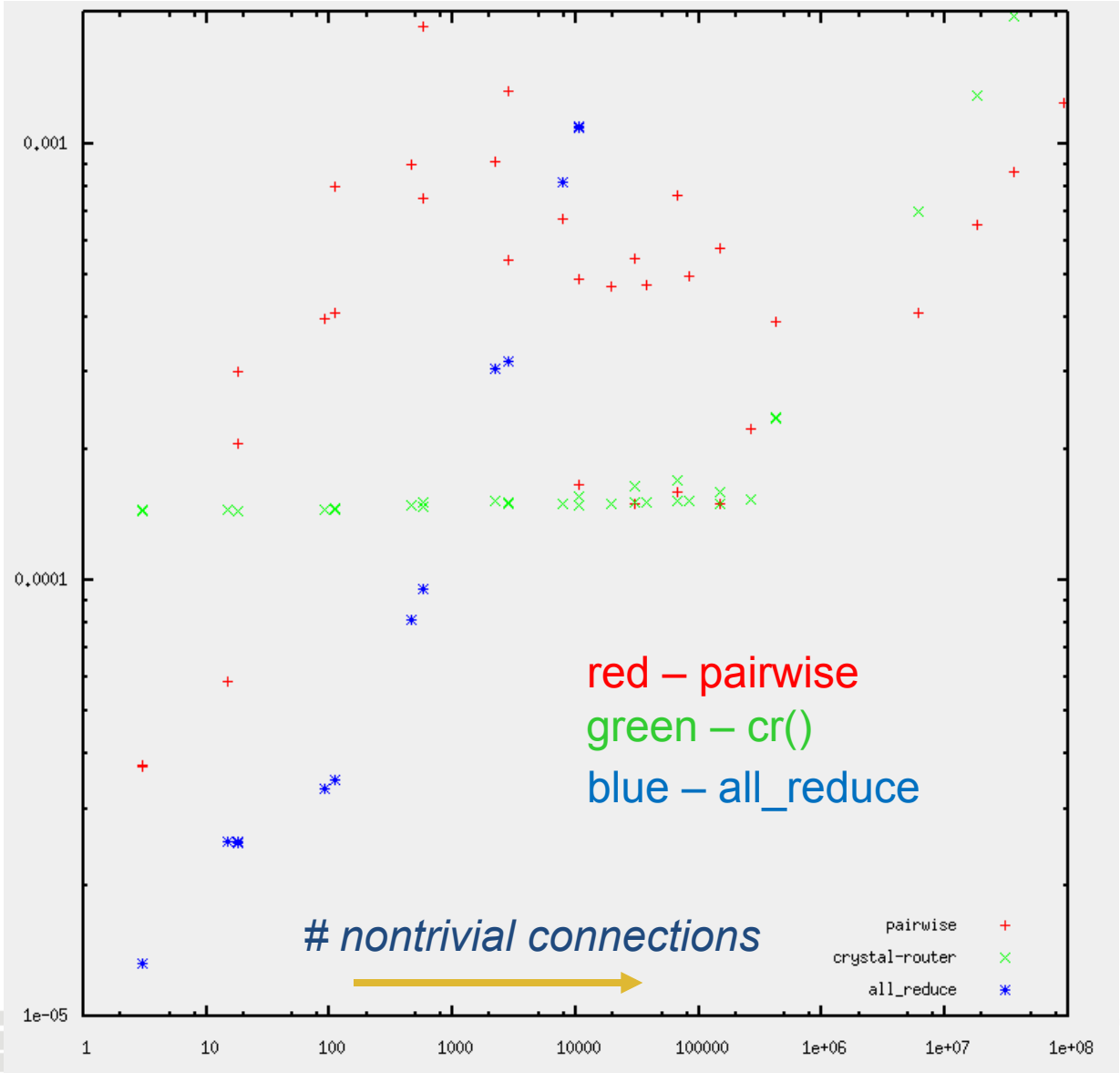
- pairwise exchange strategy is ***latency dominated*** for large stencils
- Therefore, rewrite *gs()* to switch to either crystal_router or all_reduce, if faster.

Table 3: 3D application hierarchy; $\rho_{\text{targ}} = 0.5$, $\gamma_{\text{targ}} = 0.54$

Level	n	n_c/n	$\rho_{f,m=1}$	m	γ	$\rho(E_{\text{mg}})$	$\frac{\text{nnz}(W)}{n_c}$	$\frac{\text{nnz}(\tilde{A}_{ff})}{n_f}$
1	417600	0.36	0.80	3	?	0.67	8.3	9.4
2	151248	0.44	0.61	2	?	0.63	7.9	22.0
3	66887	0.43	0.59	2	?	0.60	8.8	30.8
4	28862	0.33	0.62	2	?	0.57	12.8	41.5
5	9471	0.22	0.68	3	?	0.55	21.4	32.8
6	2116	0.18	0.67	2	0.42	0.51	36.4	86.6
7	390	0.20	0.60	2	0.42	0.48	35.8	53.8
8	79	0.16	0.72	3	0.61	0.46	33.3	43.8
9	13	0.23	0.62	2	0.39	0.35	9.3	10.0
10	3	0.33	0.44	2	0	0.11	2.0	2.0

gs() times P=131,000 on BG/P

crystal_router and all_reduce > *5-10 X* faster than pairwise in many cases



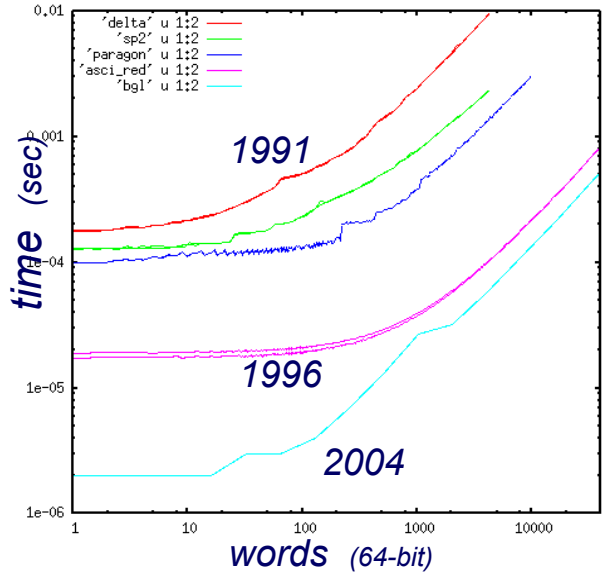
Issues

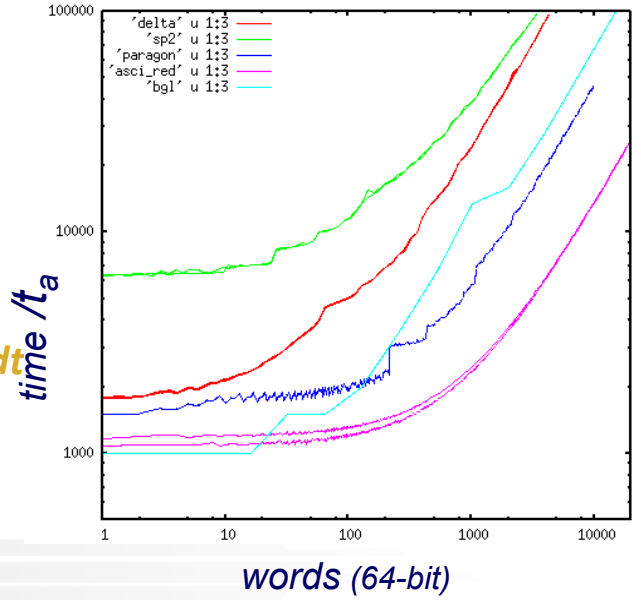
Communication Costs - 1

Linear communication model

$t_c(m) = (\alpha + \beta m)\, t_a$ **m = message size (64-bit words)**

YEAR	t_a (us)	α*	β*	α	β	m_2	MACHINE
1986	50.00	5960.	64	119.2	1.3	93	Intel iPSC-1 (286)
1987	.333	5960.	64	18060	192	93	Intel iPSC-1/VX
1988	10.00	938.	2.8	93.8	.28	335	Intel iPSC-2 (386)
1988	.250	938.	2.8	3752	11	335	Intel iPSC-2/VX
1990	.100	80.	2.8	800	28	29	Intel iPSC-i860
1991	.100	60.	.80	600	8	75	Intel Delta
1992	.066	50.	.15	758	2.3	330	Intel Paragon
1995	.020	60.	.27	3000	15	200	IBM SP2 (BU96)
1996	.016	30.	.02	1800	1.25	1500	ASCI Red 333
1998	.006	14.	.06	2300	10	230	SGI Origin 2000
1999	.005	20.	.04	4000	8	375	Cray T3E/450
2008	**.002**	**3.5.**	**.02**	**1750**	**10**	**175**	**BGP/ANL**





- $m_2 := \alpha / \beta$ ~ message size → twice cost of single-word
- t_a based on matrix-matrix products of order 10 – 13
- ***At Fine granularity, code not constrained by internode bandwidt***
- ***Latency is everything.***

Granularity Example: Standard Multigrid

- Computational complexity per V-cycle for the model problem:
  - $T_A = 50\ N/P\ t_a$
- Communication overhead:
  - $T_C := [\ (\ 8\ \alpha \log_2 N/P + 30\beta\ (N/P)^{2/3} + 8\ \alpha \log_2 P\ )\ t_a\ ]\ /\ [50\ N/P\ t_a\ ]$

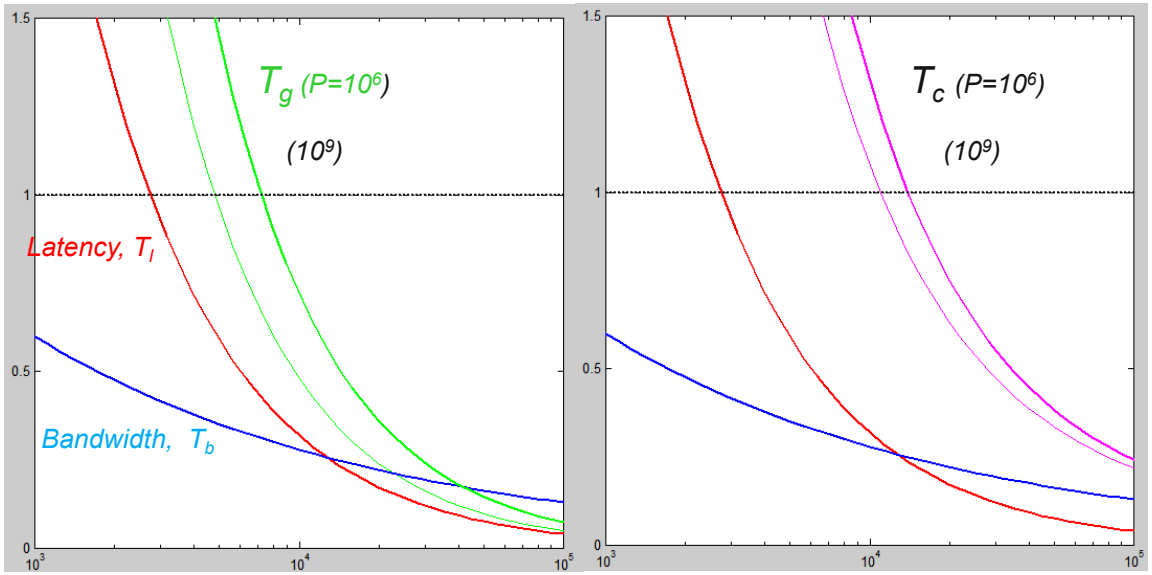
Restrict, smooth, prolongate *Coarse grid solve*

  - $T_C := T_l + T_b + T_g$ < 1 to balance communication / computation

$T_l := 8\ \alpha \log_2 N/P\ /\ (50\ N/P)$
$T_b := 30\ (N/P)^{2/3}\ /\ (50\ N/P)$
$T_g := 8\ \alpha \log_2 P\ /\ (50\ N/P)$

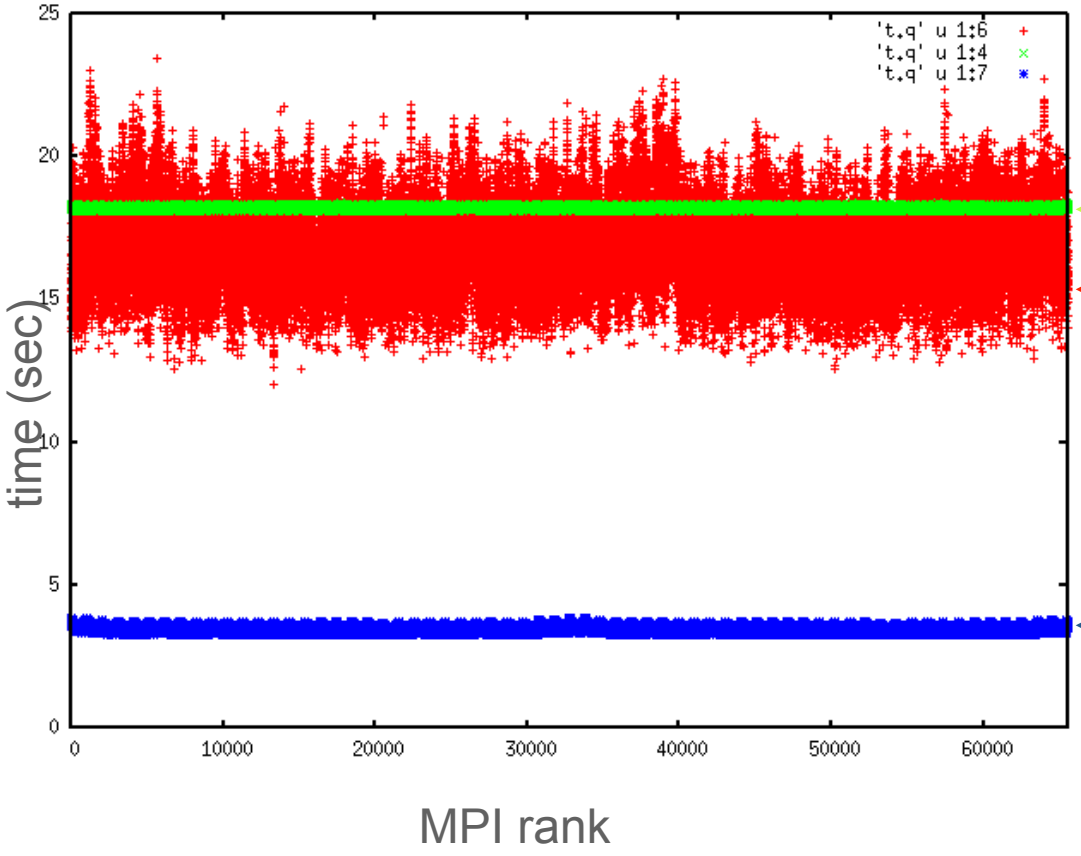
> 15000 pts/proc for $P=10^9$ (BGP)
~24 MB/proc (Nek)
> 15 trillion points total (24 PB)

- Dominated by coarse-grid solve
- Dominated by intra-node latency



Communication Costs - 2

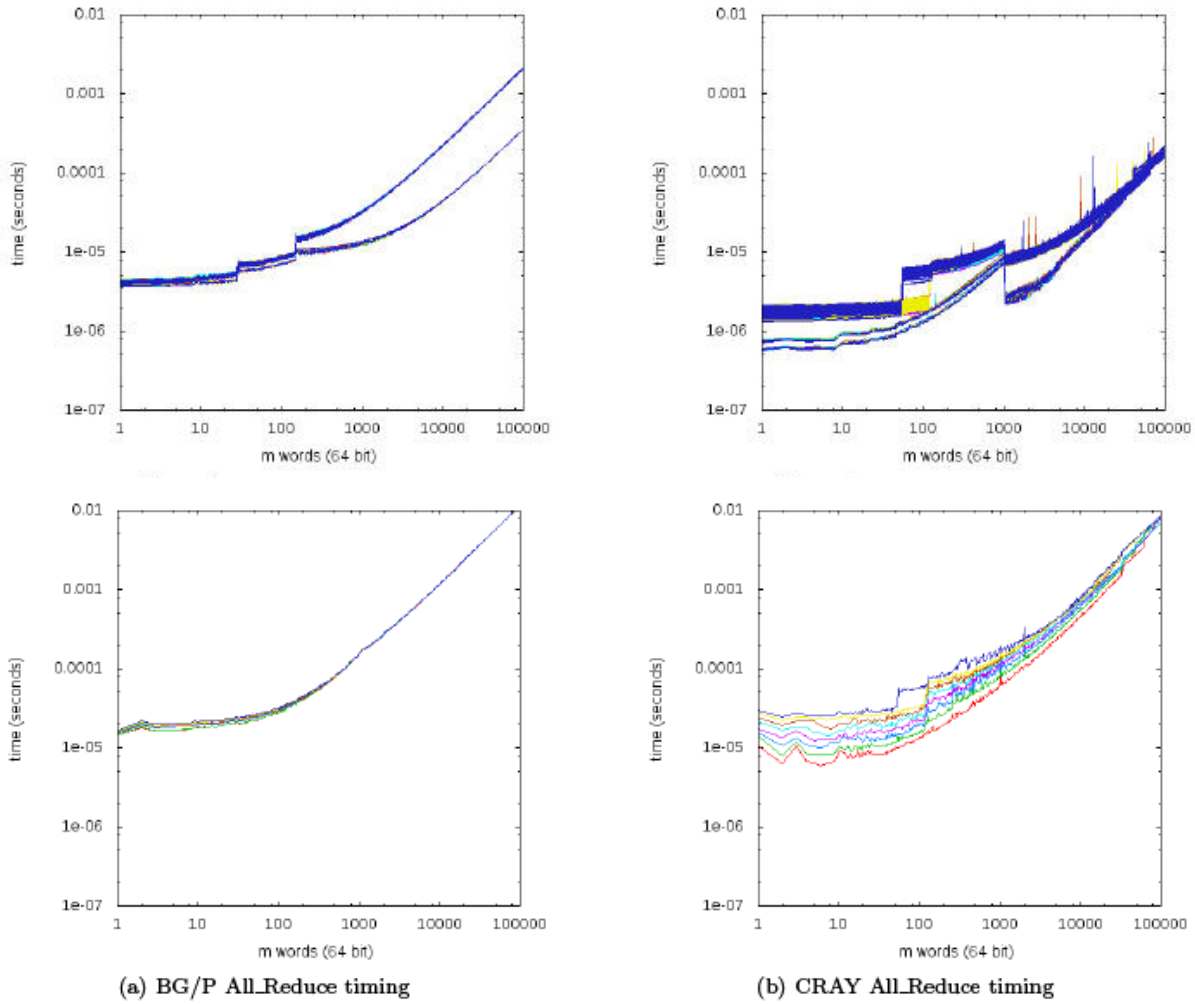
- Billion-point 217-pin bundle simulation on P=65536



- *Coarse solve time*
- *Neighbor exchange*
- *mpi_all_reduce*

- Neighbor vs. all_reduce: 50α vs 4α (4α, not 16 x 4α)

Communication Costs – 3
Raw ping-pong & all_reduce times



(a) BG/P All_Reduce timing

(b) CRAY All_Reduce timing

A major advance with BG/L and P is that all_reduce does not scale as $\alpha \log P$

How Can a User/Developer Control Communication Cost?

- Generally, one can reduce P to increase n/P
- Conversely, for a give P, what value of n will be required for good efficiency?

- Assume BGP latency, α^* = 3 usec
- Assume 100x increase in node-to-node bandwidth
- 15 x 10^6 pts/node for P=10^9 (BGP)
- Dominated by intra-node latency
- **More than 15 trillion points total ? (estimated by MG computational model)**